

Delegated Secure Sum Service for Distributed Data Mining in Multi-Cloud Settings

Dinh Tien Tuan Anh, Quach Vinh Thanh, Anwitaman Datta

Abstract—An increasing number of businesses are migrating their IT operations to the cloud. Likewise there is an increased emphasis on data analytics based on multiple datasets and sources to derive information not derivable when a dataset is mined in isolation. While ensuring security of data and computation outsourced to a third party cloud service provider is in itself challenging, supporting mash-ups and analytics of data from different parties hosted across different services is even more so. In this paper we propose a cloud-based service allowing multiple parties to perform secure multi-party secure sum computation using their clouds as delegates. Our scheme provides data privacy both from the delegates as well as from the other data owners under a *lazy-and-curious adversary* (semi-honest) model. We then describe how such a secure sum primitive may be used in various collaborative, cloud-based distributed data mining tasks (classification, association rule mining and clustering). We implement a prototype and benchmark the service, both as a stand-alone secure sum service, and as a building block for more complex analytics. The results suggest reasonable overhead and demonstrate the practicality of carrying out privacy preserved distributed analytics despite migrating (encrypted) data to possibly different and untrusted (semi-honest) cloud services.

I. INTRODUCTION

A. Multi-Party Computing Service.

An enormous amount of data is being generated everyday by a plethora of human activities and computing devices. Traditionally, data is stored in a data owner's in-house infrastructure, and access to outsiders is provided typically through web services. Example services including MedlinePlus [1], Xignite [2], NOAA [3], ResMap [4], Yahoo! Traffic [5] etc. offer a wide range of data: medical, financial market, meteorological, satellite images, traffic, etc. Data from multiple sources can be mashed-up or jointly analyzed to create new services and infer information that can not be realized from a stand alone dataset. For instance, satellite images and data from weather sensors are used together to improve forecast, financial data from multiple institutions to make better market predictions [6], traffic and human mobility data to aid urban planning [7], medical and mobility data to yield more insights on spread of diseases [8].

However, it is not always desirable or feasible to expose the data itself, and yet, being able to carry out computations or analytics over the same may provide benefits without violating the privacy concerns. Multi-party computations [9], [10] is one way to enable the same, and have recently been successfully deployed in applications such as blind auctions which require bidding information from parties to determine prices [11].

Even as multi-party computation protocols have matured to be applied in practice, for it to be widely adoptable, it is desirable to provide the same as a basic service. Our work

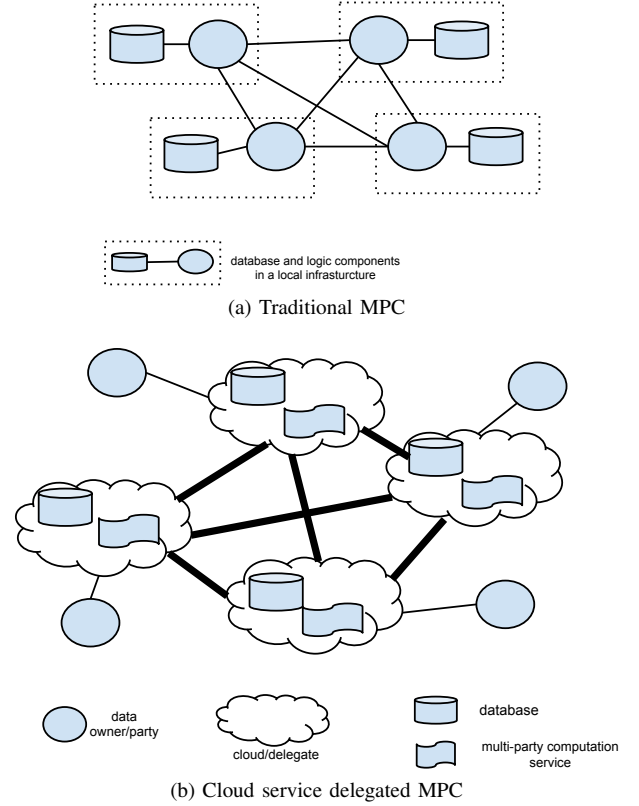


Fig. 1: Multi-party computation (MPC): In Traditional vs Multi-Cloud settings

is motivated by this observation, as well as noting another common recent trend, namely the move by many organizations to cloud based services in order to eliminate or downsize the in-house IT infrastructure.

B. Our Work.

Recent developments of cloud computing have materialized a concrete platform for rapid realization of the service-oriented computing paradigm [12]. Cloud providers offer computing as a service, from which software services can be built, sold and integrated into complex applications. Companies are leveraging the cloud for its cheap, elastic and scalable resources. Migration of IT infrastructure are taking place, in which most data, application logics and front-end services are being moved to the cloud [13]. Many existing works focus on what to migrate [13], [14], [15], assuming that the cloud is trusted. Others investigate mechanisms for protecting data privacy and for verifying computation correctness [16], [17], [18], [19], [20]. The latter works consider single-party

settings, i.e., how one data-owner can outsource its data in a privacy preserving manner, and still carry out analytics on the same. Furthermore, these works mainly explore the theoretical aspects of the problem.

Our work concerns the design space of multi-party computation outsourcing, which has so far not been explored to the best of our knowledge. In particular, we design a multi-party computation service which is invocable by authorized users when taking part in a multi-party protocol. The protocol can be run over untrusted (*curious and lazy*) clouds that act as delegates, and it guarantees individual data owners' data privacy (from the delegated clouds, as well as from other parties) and lets users verify if the computation has been carried out correctly. Specifically, we consider secure multi-party sum computation.

Figure 1 illustrates the settings of our work, contrasting it with the traditional multi-party computation setting (Figure 1a). Traditionally, each party houses its databases and computing components for handling business logics internally. To take part in a multi-party computation, they need to initiate network connections with other parties. Not only is negotiating access to internal network a troublesome endeavor [13], but also the potential heterogeneity in the network and computational resources may hinder the overall performance [21].

Our work focuses on scenarios in Figure 1b, in which parties move their data to clouds (which is in any case happening for many other reasons [12], [13]) and rely on the cloud-provided service for the multi-party computation.

Specifically, we propose a protocol allowing the cloud delegates to evaluate the sums of parties' private inputs without learning either the individual values or the resulting sums. The parties are also able to check if the sum is correct.

Not only does cloud based deployment provide the basic computation as a service, more sophisticated analytics services can be realized using it as building block. We demonstrate this by designing some representative data mining tasks such as Naive Bayes (classification), Apriori (association rule mining) and K-Means (clustering). The resulting protocols support encrypted databases, hence the cloud delegates cannot learn the private data. Naturally, there are overheads in using the service, as a trade-off for the security guarantees. However, it is amortized when used within complex applications.

Contributions. Our contributions are as follows:

- 1) We present a multi-party computation (secure sum) service which can be run on curious-and-lazy cloud delegates while maintaining data privacy and correctness. The service can work with encrypted databases, and it ensures computation correctness with a minimal coordination among the parties outsourcing the task.
- 2) We demonstrate how this service can be used in complex cloud-based data mining jobs.
- 3) We benchmark the secure sum service as a stand-alone application in a multi-cloud like environment. We also experiment with various data mining applications using our services. Compared to the traditional multi-party computation implementations without delegates (we refer to them as the non-delegated versions), there are naturally overheads due to the added security mechanisms.

However, the experiments show that such overheads are within reasonable range for applying our approach in practice, and the cost amortization becomes more prominent with increased analytics workloads.

Organization. The rest of this paper is structured as follows: Section II details the delegated secure sum protocol in an abstract manner. Section III shows how complex data mining tasks can be built with this service. We consider three classic data mining algorithms: classification, association rule mining and clustering. Section IV follows with experimental benchmarkings of the protocol and the data-mining tasks. Related works are discussed in Section V before we draw our conclusions and outline the future directions for this work in Section VI.

II. DELEGATED SECURE SUM

We briefly explain two traditional approaches for multi-party secure sum computation, based upon one of which our service is built. We suppose a number of parties P_0, P_1, \dots with private inputs x_0, x_1, \dots wish to compute $s = \sum_i x_i$ in a privacy-preserving manner, so that P_i will not learn x_j for $j \neq i$.

In the *ring-based* approach [22], the parties form a ring and messages are forwarded in a pre-defined direction. A *master party* P_m is elected and starts the computation by sending $v = x_m + r$ (for a random r) to its immediate clockwise (or counter-clockwise) neighbor, which adds its own input to v and forwards the result along. Once arrived back at the master party, the final sum is obtained as $s = v - r$ and broadcast to other parties.

Another approach is based on broadcast communication [23], [24]. At the beginning, each party generates r_i such that $\sum_i r_i = 0$. Next, P_i sends $v_i = x_i + r_i$ to other parties, and independently computes the sum $s = \sum_i v_i$. Our work is an extension of this broadcast-based protocol. We discuss the trade-offs between broadcast and ring-based protocols in Section II-D which prompted us to make the specific choice.

A. Model

Our model consists of a number of **parties** and **delegates**. The party P_i sends its transformed input $\phi(x_i)$ to its delegate D_i , and receives s at the end. The delegates D_0, \dots, D_{n-1} are connected to each other. They receive inputs from the parties, start a multi-party computation and finally send back the results.

1) *Adversary model.*: We assume that delegates are *curious and lazy*. They will try to learn inputs from the parties and/or the resulting sum. To that end, they may listen to the communication channels, but will not try more active attacks that are to deliberately subvert the computation. For example, we do not consider attacks in which an adversarial delegate sends different values in a computation round in order to partition the result. However, we assume that delegates may have incentives to be lazy, i.e. to do as little as possible (and still charge the parties). Specifically, they may skip some (or all of the) computations, replay results from previous rounds, or even replace party inputs with other values to save computation

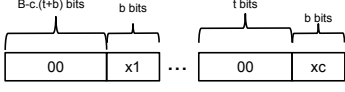


Fig. 2: Packed Paillier scheme. c plaintext values are packed into a single plaintext of B bits

time, which could consequently compromise the integrity of the sum computation.¹

We assume that every party/data-owner is *honest-but-curious*, i.e., a data owner will not diverge from the protocol but may try to learn other parties' inputs. This is a standard assumption often used in designing multi-party computation protocols, where the resulting approaches are simple yet secure, even though they do not deal with attack scenarios in which adversaries are in majority with respect to the honest parties.

Dishonest delegates may collude with each other, and also with the parties. However, the party-delegate *collusion is weak*, in the sense that the party can ask for messages seen by the delegate, but the party would not reveal its secret keys. Since all the parties are legitimate recipients of the final answer, if a party colludes to the extent of providing a delegate the decryption key or the final outcome - then naturally, this (or any other) protocol can not safeguard against such a situation.

2) *Crypto model.*: Each party has a public key pair (PKp_i, PKs_i) , each delegate has a pair (DKp_i, DKs_i) . We assume that parties are running a common, well-known random number generator (RNG). Our protocols use an *additively homomorphic* encryption scheme, whose encryption $\text{Enc}(eK, m)$ and decryption $\text{Dec}(dK, c)$ operations satisfy:

$$\text{Enc}(eK, m_1) \oplus \text{Enc}(eK, m_2) = \text{Enc}(dK, m_1 + m_2)$$

for an operation \oplus . ECC-Elgamal [25] and Paillier [26] are two of such schemes, both providing randomized encryptions. ECC-Elgamal uses additive groups of an elliptic curve, whereas Paillier relies on composite residuosity classes over a group. The former is more efficient, but requires homomorphic transformations of plaintexts to and from an elliptic curve, which is expensive. Our work uses the latter. It requires larger bit-length, but we can employ an optimization that allows us to perform multiple encryptions at the same time. Specifically, we pack multiple values in one plaintext so that they can be encrypted at the same time [16]. Suppose plaintext values are at most b bits and Paillier's plaintexts are B bits. Suppose further that any sum value is smaller than 2^{t+b} , then we can pack c values into a single plaintext, as demonstrated in Figure. 2, where $c \leq \lceil \frac{B}{b+t} \rceil$. Denote

$$pm(\bar{v}) = 0 \| x_1 \| 0 \| x_2 \dots 0 \| x_c$$

as the packed Paillier message using values $\bar{v} = \langle x_1, x_2, \dots, x_c \rangle$. We can extract x_i ($1 \leq i \leq c$) as: $x_i = pm(\bar{v})[i] = \lfloor pm(\bar{v}) \gg ((c-i) * t + b) \rfloor \& (2^{t+b} - 1)$. The

¹This simplified adversary model is justified by legal and economic realities, where a commercial cloud service provider may try to over-charge customers or try to (passively) sniff information readily exposed without the liability of committing a criminal offense, as opposed to launching any deliberate (proactive) attack to subvert confidentiality.

homomorphic property is maintained, i.e.:

$$\begin{aligned} cipher &= \text{Enc}(eK, pm(\bar{v})) \cdot \text{Enc}(eK, pm(\bar{v}')) \\ &= \text{Enc}(eK, pm(\bar{v} + \bar{v}')) \end{aligned}$$

B. Delegated Secure Sum Protocol

The protocol comprises four phases: SETUP, ENCRYPT, COMPUTE and VERIFY. The SETUP phase is run once to initialize the system and security parameters, the rest are needed for each computation round.

SETUP. The goal of this phase is two-fold. First, parties agree on a secret Paillier key pair (eK, dK) , and an initial value `roundId`. Although Paillier is a public-key scheme, we treat it as a symmetric scheme, where both public and private key are kept secret. Second, each party generates a random value r_i (later used for perturbing its inputs) such that $\sum_i r_i = 0$ (explained shortly).

To generate (eK, dK) and `roundId`, the parties first agree on a secret X , then use it as the seed for the RNG. Having the same source of randomness, they run the same algorithms for creating (eK, dK) and `roundId`. One method for establishing X could be to appoint a master party which generates X and distributes it to the rest. This way, the master produces and sends $O(N)$ different ciphertexts to other parties. We instead adopt a cryptographic approach first proposed for secret key exchange [27], in which every party contributes its own randomness to the final value. In our context, each party sends only 2 messages, while most of the computations can be outsourced to the delegates. Computations are done in a globally known group of prime order $G_p(g)$:

- 1) P_i generates a random value x_i and sends $z_i = g^{x_i}$ to D_i .
- 2) D_i broadcasts z_i , then forwards $a = (z_{i+1}/z_{i-1})$ once it has received from other delegates.
- 3) P_i computes $X_i = a^{x_i}$ and sends to D_i .
- 4) D_i broadcasts X_i , then forwards $b = z_{i-1}^n$ and $c = X_i^{n-1} \cdot X_{i+1}^{n-2} \dots X_{i-2}$ to P_i .
- 5) P_i computes $X = b^{x_1} \cdot c = g^{x_1 \cdot x_2 + \dots + x_n \cdot x_1}$, which is the shared secret seed between all parties.

Having generated (eK, dK) , P_i constructs $\text{Enc}(eK, i \| 0)$ and sends it to D_i which broadcasts to other D_j and consequently to party P_j . On receipt of m_j for all $j \neq i$, P_i decrypts it and checks if the message was properly formed. If successful, it means that X and (eK, dK) have been agreed upon by all parties. Each assigns the next random number from the RNG as `roundId`.

A common RNG cannot be utilized to create r_i satisfying $\sum_i r_i = 0$. Instead, we adopt an approach proposed in [24]. Specifically:

- 1) P_i generates random values r_{ij} for $i \neq j$.
- 2) P_i sends $m_{ij} = \text{Enc}(PKp_j, r_{ij})$, $\text{sign}_i(m_{ij})$ for all $j \neq i$ to D_i , which then distributes it to D_j and subsequently to P_j .
- 3) P_i decrypts and verifies signatures of m_{ji} ($j \neq i$). Then it computes:

$$r_i = \sum_{j \neq i} (r_{ij} - r_{ji})$$

It can be seen that $\sum_i r_i = \sum_i \sum_{j \neq i} (r_{ij} - r_{ji}) = 0$.

ENCRYPT. In this phase, each party encrypts its private input x_i and sends to the delegate. More specifically, P_i packs roundId and $x_i + r_i$ into a single plaintext and encrypts it with Paillier, the result is $m_i = \text{Enc}(\text{eK}, \text{roundId} || (x_i + r_i))$. When there are multiple inputs $\{x_{i0}, x_{i1}, \dots, x_{ik}\}$, P_i packs them into a smaller number of plaintext m_{i0}, m_{i1}, \dots with increasing values of roundId .

COMPUTE. On receipt of m_i from party P_i , the delegate D_i broadcasts this value to other delegates. Once having all messages from other delegates, it computes

$$\begin{aligned} c &= \prod_{0 \leq i < n} m_i = \prod_{0 \leq i < n} \text{Enc}(\text{eK}, \text{roundId} || (x_i + r_i)) \\ &= \text{Enc}(\text{eK}, n.\text{roundId} || \sum_i x_i) \end{aligned}$$

and sends it back to the party. Each delegate also signs the result message: $\sigma_i = \text{sign}_i(i || c)$.

VERIFY. Once receiving c , P_i asks D_i for its signature as well as signatures from t other delegates. P_i can decide at random the value of t and identities of the verifying delegates. For instance, it can set $t = 2$ and ask its delegate for the signatures from D_i, D_{i-1}, D_{i+1} : $\sigma_i, \sigma_{i-1}, \sigma_{i+1}$. Once successfully verifying that the signatures are correct, P_i decrypts it to get:

$$s = \text{Dec}(\text{dK}, c) = n.\text{roundId} || \sum_i x_i$$

Finally, it extracts $s[0], s[1]$ from s , and checks that $s[0] = n.\text{roundId}$ before assigning $s[1]$ as the final sum value.

C. Security Analysis

Our protocol has the following security properties. First, curious delegates and parties cannot see private inputs of the honest parties, even if they collude with each other. This is because each input x_i is randomized with r_i . Since the generation of r_i is done in a secure manner, only P_i will be able to recover x_i from $(x_i + r_i)$.

Second, delegates cannot see the sum value $\sum_i x_i$. It has the encrypted value $s = \text{Enc}(\text{eK}, n.\text{roundId} || \sum_i x_i)$, but cannot extract the sum since it has no knowledge of the decryption key. Recall that our model does not consider active collusion between dishonest parties and delegates in which secret keys are revealed to the delegates.

Third, the protocol computes the correct sum from the party inputs, provided that at least one of the t verifying delegates is honest. This means dishonest delegates cannot skip computations or replay old values. They cannot also replace party inputs with other values without getting caught. The sketch of proof is as follows. Since the delegate does not know the encryption key, should it use an input different to what is given from the party, the VERIFY step will fail because $s[1] \neq n.\text{roundId}$. It cannot replay old values either, as each round is *tagged* with a unique value of roundId . The delegate can skip the computation in two ways. First, it may not use its party's input m_i during the COMPUTE step. However, this causes the VERIFY step to fail since $s[1] \neq n.\text{roundId}$. Second, it may ignore m_j from other

delegates. In this case, to ensure $s[1] = n.\text{roundId}$, the delegate must construct $s = (m_i)^n$ (raising m_i to power of n is cheaper than multiplying n different values). However, the VERIFY step also checks for the results from other delegates, of which one is honest and therefore its result must be different to s . Hence, the verification will again detect D_i 's laziness.

These security guarantees are achieved while assuming that the cryptographic schemes are secure. Attacks on these primitives will inevitably affect our protocols, but they are outside the scope of this paper. The VERIFY step is done at the end of every round, which can be expensive over many rounds. We extend the protocol to support probabilistic verification, in which verification is carried out with a probability p at any given round. The probability of detecting misbehavior can be made arbitrarily high after a number of verifications. Specifically, let ck be the number of random checks, $P(p, n, ck)$ be the probability that misbehavior is detected after ck checks (assuming that delegates misbehave consistently²). Then:

$$P(p, n, ck) = 1 - (1 - p)^{n \cdot ck}$$

D. Broadcast vs Ring-based delegated secure sum

The traditional ring-based protocol [22] can be extended to support delegates as follows. A party P_i still encrypts $(x_i + r_i)$ in the same way using the shared Paillier key, and sends it to D_i . The master delegate D_0 starts the computation by forwarding m_0 along the pre-defined direction, with which each delegate on the path multiplies its party input and sends the result along the same direction. Once a message c arrives at D_0 , it was broadcast to other delegates and subsequently to their parties. Verification at parties involves performing the same checks as described above. This simple protocol ensures that delegates cannot learn private inputs and the sum outputs, however it is possible for lazy delegates to skip computations and render the sum value incorrect. For example, suppose D_0 sends m_0 to D_1 which then computes $c_1 = m_0.m_1$ and forwards to D_2 . Suppose D_2 and D_3 are both dishonest, they can bypass the computation $c_2 = c_1.m_2$ and $c_3 = c_2.m_3$ by sending $c_3 = c_1.c_1$ to D_4 . The finally verification checks out, but the sum is not correct.

To achieve correctness, several enhancements are needed. The master party first splits the value $v_0 = x_0 + r_0$ into two parts v_0^l and v_0^r such that $v_0^l + v_0^r = v_0$. D_0 then sends m_0^l to its left neighbor and m_0^r to its right neighbor (m_0^l, m_0^r are the ciphertexts of v_0^l, v_0^r). Effectively, there are two streams of messages in opposite directions. Party P_i receiving σ_{in}^l and σ_{in}^r from its two neighbors, it computes $\sigma_{out}^l = \sigma_{in}^l.m_i$, $\sigma_{out}^r = \sigma_{in}^r.m_i$ and forwards them to the next neighbor. Two messages arriving at the master delegates are σ^l and σ^r . During verification, party P_i checks the following conditions:

- 1) $\text{Dec}(\sigma^l) = \text{Dec}(\sigma^r)$ and $\text{Dec}(\sigma^l)[1] = n.\text{roundId}$.
- 2) $\text{Dec}(\sigma_{in}^l)[1] = i.\text{roundId}$ and $\text{Dec}(\sigma_{in}^r)[1] = (n - i).\text{roundId}$ and $\text{Dec}(\sigma_{in}^l) + v_i = \text{Dec}(\sigma_{out}^l)$ and $\text{Dec}(\sigma_{in}^r) + v_i = \text{Dec}(\sigma_{out}^r)$.

²If the delegate misbehaves probabilistically with some probability p' , then the detection probability will be $1 - (1 - p \times p')^{n \cdot ck}$

$$3) \text{Dec}(\sigma_{in}^l)[2] + \text{Dec}(\sigma_{out}^r)[2] = \text{Dec}(\sigma_{in}^r)[2] + \text{Dec}(\sigma_{out}^l)[2] = \text{Dec}(\sigma^l)[2]$$

This protocol's only advantage over our broadcast based design is that each delegate maintains two connections to its immediate neighbors. Therefore, the communication cost for the delegates are constant, whereas in our protocol each delegate must handle $O(n)$ messages. However, there are many added overheads. First, the master party has to wait until the message traverse the ring completely before broadcasting the result. This does not scale with an increasing number of delegates, as network latency will become significant. Second, parties need to establish other shared states besides Paillier keys. Particularly, they have to agree on the identity of the master party, and on which direction to forward the messages. In case of probabilistic verification, they also must agree on which rounds to carry out the checks. Having many shared states makes the system less robust in presence of failure. Finally, electing a node as a master party to whom the computation correctness is rested requires such trust assumption that may not be realistic in decentralized settings.

III. DATA MINING APPLICATIONS

A data mining algorithm can be modeled as a process of extracting knowledge from a database, which consists of two iterative steps: $\bar{x} \leftarrow \text{query}(D)$ that queries a database D , and $\text{process}(\bar{x})$ that computes knowledge from the query results. In a distributed settings, the first step is executed locally, and the results are aggregated across multiple parties. In other words, a distributed data mining algorithm now comprises three steps: $\bar{x}_i \leftarrow \text{query}(D_i)$, $\bar{x} \leftarrow \text{aggregate}(\bar{x}_i)$ and $\text{process}(\bar{x})$. Possible aggregate functions are: sum, set union, scalar product, etc. [28]. Our protocol described in the previous section has provided an implementation for the sum function. We now take three classical data mining tasks (classification, association rule mining and clustering) and show how they can be realized utilizing our service in a collaborative, cloud-based settings.

A. Secure Database Outsourcing

Database outsourcing has always been an attractive option for small-to-medium businesses even before the era of cloud computing. Main reasons for moving data to third-party sites are: scalability, high-availability and cost effectiveness, freeing up an enterprise's resources for its core business priorities. With the advent of cloud computing, this trend has tremendously accelerated.

For simplicity, we assume that databases are in relational format, and every attribute has a non-negative integer domain (other domains could be mapped into the integer domain, the details of which are not within the scope of our work). The party is the data owner, who wishes to move the data and computation to the cloud (or delegate). We assume that the party retains a local copy, but most queries and processing on the data are to be done on the cloud. This may be the case, for instance due to the availability of arbitrary amount of computing power and specialized tools (software services) on the cloud, which are hard to achieve in-house.

That data residing at a third-party and queries being executed remotely raise several security issues: data privacy, integrity, query completeness and query freshness. Since our adversary model for the cloud is curious and lazy (Section II-A), we will only deal with the data privacy and query completeness problems. Techniques for ensuring query freshness are discussed elsewhere [29], [30].

1) *Data privacy.*: To protect data privacy, encryptions can be used. Many encryption schemes exists, each differs to another in its security guarantee and the range of operations that can be done over the ciphertexts. When outsourcing data, the trade-off between security and possibility of computing over ciphertexts must be made in order for queries to be executable at a third-party [16]. Randomized encryptions (for example AES in CBC mode with randomized initialization vector) offer security against adaptive chosen-plaintext attacks, but no meaningful computations can be done. Deterministic encryptions (DETs) such as AES offers less security, but facilitates equality comparison: $\text{Enc}(x) = \text{Enc}(y) \leftrightarrow x = y$. Order-preserving encryptions (OPEs) [31], [32] support inequality comparisons on ciphertexts: $\text{Enc}(x) < \text{Enc}(y) \leftrightarrow x < y$. However, they have weaker security guarantee than DETs, since they reveal plaintext's order. Another useful encryption primitive is homomorphic encryptions (HOMs) such as the Paillier scheme used in our secure sum protocol. Such schemes are inherently malleable. Using DETs, simple queries such as equality selection, COUNT and GROUP BY can be performed by database engines. With OPEs, MIN, MAX, SORT, ORDER BY are also supported. Furthermore, these can be done efficiently since the database engine can build its B+ tree directly from the ciphertexts. With HOMs, aggregate functions like SUM or AVG can be performed.

Since there are different types of queries, multiple types of encryptions must be supported at the same time. In CryptDb [16], data is encrypted in multiple *onions* used for different use-cases, each onion is multi-layer: the outer-most layer is the most secure and the inner-most supports the most complex operations. The database queries required for the data mining algorithms in our work are limited to those supported by DETs and OPEs. We adopt a simple approach that stores two copies of the database at the cloud: one encrypted with AES and one with OPE. Column names and table names are also encrypted with AES. A database query is transformed to the encrypted version, by encrypting column name, table name and attribute values with the appropriate encryption key and scheme. For example, two queries:

```
select COUNT from t
  where t.a1 = x AND t.a2 = y;
```

```
select COUNT from t ORDERED BY t.a1
  where t.a1 < x AND t.a2 < y;
```

are translated into:

```
select COUNT from t_aes
  where t_aes.AES(a1) = AES(x)
     AND t_aes.AES(a2) = AES(y);
```

```

select COUNT from t_ope
  where t_ope.AES(a1) < OPE(x)
    AND t_ope.AES(a2) < OPE(y)
  ORDERED BY t_ope.AES(a1)

```

where t_{aes} and t_{ope} are the encrypted names of the AES-encrypted and OPE-encrypted table derived from the original table t . We assume that both database encryption and query translation is done by the data owner (or by authorized users).

2) *Query completeness.*: Executing a query over a large database is an expensive operation. As the cloud is lazy, it has incentives to skip some parts of the data when performing the query, or even ignore the query altogether and return a random response instead. Li et al. proposed a datastructure called Authenticated Aggregation R tree (AAR tree [33]) that can produce a proof that the query has been executed over the complete data. However, the most complex queries that AAR tree can support is COUNT or SUM over range selection conditions, plus the proofs are expensive to construct and verify.

We propose a probabilistic solution that takes advantage of the data owner having a local copy of the database. A straight-forward protocol would require the data owner to probabilistically execute a query q over its local copy of the data and compare the result with what returned from the cloud. Thus, let p be the probability that the cloud is lazy for any given query, then the probability of it getting caught after k checks will be $1 - (1-p)^k$, whose value rapidly approaches 1. Notice that even though k may be small, executing q directly on the local database may not be desirable, as the party wishes to be involved as little as possible.

We observe that even if the cloud ignores some parts of the data, the data mining output may still be accurate. We conduct experiments with NaiveBayes and K-Means algorithms to test the accuracy when blocks of data are removed at random. We divide the data into b blocks, and use mis-classification rates and root mean square errors as accuracy metrics. The result depicted in Figure 3[b] suggests that accuracy indeed remains high. The output of NaiveBayes, for example, is above 99.6% correct even when 30% of the blocks are removed. Another observation is that when data is divided into blocks and the cloud removes a considerable number of blocks, the party may execute the query only over a small number of blocks in order to detect inconsistency. Our protocol is based on [34], and proceeds as follows. The query q is transformed to a list of smaller queries $Q = \langle q_1, q_2, \dots, q_b \rangle$, each executed on one block. The party sends Q to the cloud. Let r, w be the number of queries in Q performed by the party locally and by the cloud at its site. The probability $P(b, w, r)$ that the cloud stays undetected when performing only w out of b queries are:

$$P(b, r, w) = \frac{1}{\binom{b}{r}} \sum_{i=\max(0, w+r-b)}^{\min(r, w)} \frac{\binom{w}{i}}{\min(b-w, \max(1, b-i))}$$

Figure 3[a] shows that the probability of successful cheating decreases exponentially when the cloud ignores more data. It indicates that the party only needs to execute the query over a small portion of its local data (10 – 15%) for the detection to be effective. This suggests that the party may not need to

store the entire database locally: it may suffice to have 20% of the data (unknown to the cloud delegate) and refresh them at pre-defined intervals. We leave further investigation to this question for future work.

In summary, our protocol is effective for ensuring query completeness: if a large amount of data is ignored, the cloud gets caught with very high probability; if a small part is ignored, the data mining algorithms are largely unaffected.

B. Data Mining Algorithms

Algorithm 1: Naive Bayes classification

Input: Y, A, V , local party p
Output: $N, N_y, N_{y,a,v}$ for all y, a, v

```

1 foreach  $y \in Y$  do
2    $N_y^p \leftarrow \text{QueryCount}(l_x = y)$ 
3   foreach attribute  $a \in A$  and  $v \in V_a$  do
4      $N_{y,a,v}^p \leftarrow \text{QueryCount}(x_a = v, l_x = y)$ 

5 foreach  $y \in Y$  do
6    $N_y \leftarrow \text{secursum}(N_y^p)$ 
7   foreach  $a \in A$  and  $v \in V_a$  do
8      $N_{y,a,v} \leftarrow \text{secursum}(N_{y,a,v}^p)$ 

```

1) *Classification (Naive Bayes)*: A classification algorithm takes as input a set of labeled (training) data and outputs a *classifier* that can be used to label new (test) data. Let Y be a set of labels, $l_x \in Y$ be the label of x (a multi-variate vector). Let $A = \{a_1, a_2, \dots\}$ be a set of attributes (columns) and $V = \{V_i\}$ be the set of attribute domains. Let N be the number of instances (rows), N_y be the number of instances (rows) with label y , $N_{y,a,v}$ be the number of instances whose column a 's value is v and whose label is y . The NaiveBayes algorithm (Algorithm 1) returns $(N, N_y, N_{y,a,v})$ for all y, a, v as the classifier [35]. The label of a new instance x is: $\text{argmax}_y (\frac{N_y}{N} \cdot \prod_i \frac{N_{y,i,x_i}}{N_y})$.

$\text{QueryCount}(x_1 = v_1, \dots)$ follows the protocol for querying outsourced databases described in the previous section. Basically, the party constructs a COUNT query of the form

```

select COUNT(.) from Data
  where (x_1=v_1) AND ..

```

, then encrypts the names $Data, v_1, v_2, \dots$ before sending it to the cloud. The cloud executes the query, returns a result which is verified (if applied). $\text{secursum}(x)$ invokes the secure sum service using x as party input. If verification of the query or secure sum process fails, the algorithm terminates.

2) *Clustering (K-Means)*: Clustering algorithms partition data into separate *clusters* such that distance between elements belonging to the same cluster is smaller than between ones in different clusters. The K-Means algorithm (Algorithm 2) finds k clusters identified by their *centroids* (the mean centers of the clusters). Each party starts with a set of chosen centroids, then computes new centroids by grouping data into clusters using previous centroids. The algorithm works in multiple rounds until convergence (i.e., the set of centroids is unchanged compared to the previous round). This algorithm is slightly different from the standard K-Means [36], for we are only

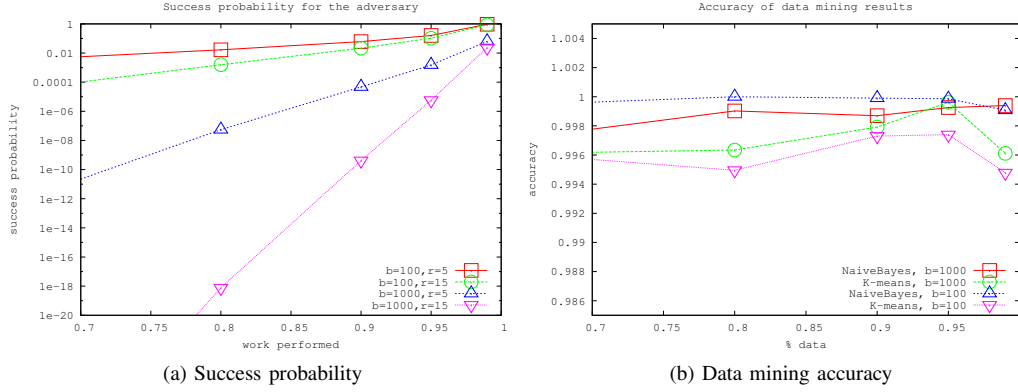


Fig. 3: Query completeness with probabilistic verification

Algorithm 2: K-Means Clustering

Input: Number of clusters k , A , local party p
Output: Set of centroids $M = \{m_1, \dots, m_k\}$

```

1 foreach  $m_i \in M$  do
2    $m_i = (i, i, \dots, i)$ 
3  $C^p = \emptyset$  foreach  $m_i \in M$  do
4    $C_{m_i}^p = \emptyset$ 
5   foreach  $a \in A$  do
6      $C_{m_i}^p(a) \leftarrow \text{QueryGroupBy}(a, m_i, M)$ 
7      $C_{m_i}^p = C_{m_i}^p \cup C_{m_i}^p(a)$ 
8    $C^p = C^p \cup C_{m_i}^p$ 
9 foreach  $C^p[i] \in C^p$  do
10   $C[i] \leftarrow \text{secursum}(C^p[i])$ 
11  $C^p \leftarrow C$ 
12 foreach  $m_i \in M$  do
13   Extract  $C_{m_i}^p$  from  $C^p$ 
14   foreach  $a \in A$  do
15      $m_i(a) \leftarrow \text{Mode}(C_{m_i}^p(a))$ 
16 Repeat Step 3  $M$  converges.

```

dealing with integer-domain attributes (or categorical data). Specifically, *data mode* metric (the most frequently seen value, computed by the Mode function) is used instead of mean.

Many different metrics exist for quantifying distance from an element x to a centroid c . We use Manhattan distance for its simplicity. In particular: $\Delta(x, c) = \sum_i |x_i - c_i|$. $\text{QueryGroupBy}(a, m_i, M)$ asks the cloud to return a list of frequencies for attribute a in the portion of data closest to the centroid m_i . The database query has the following form:

```

select a, COUNT(*) from Data as freq
where  $\Delta(a, m_i) < \Delta(a, m_0)$  AND ...
      AND  $\Delta(a, m_i) < \Delta(a, m_{i-1})$ 
      AND  $\Delta(a, m_i) < \Delta(a, m_{i+1})$  AND ...
Group by a, Order by a

```

As Δ is computed over OPE ciphertext, what returned from the cloud for QueryGroupBy might not be correct. OPE's only guarantee is $\text{Enc}(x) < \text{Enc}(y) \leftrightarrow x < y$, hence it does not always follow that $|\text{Enc}(x) - \text{Enc}(x')| < |\text{Enc}(y) - \text{Enc}(y')| \leftrightarrow |x - x'| < |y - y'|$. Suppose that in the unencrypted database, an element x is closer to centroid c_1 than c_2 , the distance based on OPE values of x , c_1 and c_2

might indicate that x is closer to c_2 . Our experimental study (discussed later) shows that the phenomenon occurs frequently, but the final clusters are very close to the clusters found using the unencrypted databases.

Algorithm 3: Apriori association rule mining

Input: Support threshold minsup and confidence threshold minconf
Output: Set of association rules $X \rightarrow Y$

```

1  $L_1 \leftarrow \text{GenerateFrequentItemsetSize1}()$ 
2  $k = 2$ 
3  $C_k \leftarrow \text{GenerateCandidates}(L_{k-1})$ 
4  $B_p = \emptyset$ 
5 foreach  $c \in C_k$  do
6    $t \leftarrow \text{QueryCount}(c)$ 
7    $B_p = B_p \cup t$ 
8 foreach  $1 \leq i \leq |C_k|$  do
9    $B[i] \leftarrow \text{secursum}(B_p[i])$ 
10  Extract  $c.\text{count}$  from  $B[i]$ 
11  $L_k \leftarrow \{c \in C_k \mid c.\text{count} \geq \text{minsup}\}$ 
12 Increase  $k$  and repeat from line 3 until  $L_k = \emptyset$ 
13  $\text{GenerateRules}(\bigcup_k L_k, \text{minconf})$ 

```

3) *Association rule mining (Apriori)*: Association rule mining algorithms extract relationships between attributes that occur frequently in the data. An association rule is of the form $X \rightarrow Y$ where $X, Y \subseteq A$. Apriori algorithm (Algorithm 3) first determines frequent item sets containing a single item. $\text{GenerateFrequentItemsetSize1}$ issues COUNT queries with different attributes. The results are merged into a set of candidates (larger item sets, using $\text{GenerateCandidates}$). The threshold value minsup specifies the lower bound for item set frequency. These steps are repeated until there is no more item set to be found. Finally, GenerateRules generates outputs by establishing rules between non-empty subsets and removing rules whose confidence values are below minconf . The details of $\text{GenerateFrequentItemsetSize1}$, $\text{GenerateCandidates}$ and GenerateRules can be found in [37].

	4 delegates
encryption (512-bit)	0.64 (± 0.15)
decryption (512-bit)	0.36 (± 0.11)
encryption (1024-bit)	4.53 (± 0.18)
decryption (1024-bit)	2.49 (± 0.17)
signing (1024-bit RSA)	1.01 (± 0.005)
signature verification	0.38 (± 0.05)

Fig. 6: Cost of cryptographic operations (ms)

IV. EVALUATION

A. Prototypes

We have implemented a prototype for the protocols discussed in the previous sections³. The secure sum service is implemented in Java, in which AES encryptions and RSA signatures are provided by the Crypto++ library [38], OPE and Paillier encryptions by CryptDB library [16]. Communications between parties and delegates are done via Java sockets, using the thread-per-connection model. The data mining algorithms are implemented in Java, using the Weka library [39]

We now discuss our experiments for evaluating the secure sum service, first as a stand-alone service, then as being a part of complex data mining applications. All experiments are run in a cluster of 16 nodes, each has a Xeon processor 3.0Ghz, running OCS5.1 (2.6.18-53El5smp) operation system with 4GB of RAM. The machines are connected via InfiniBand 20Gbps.

B. Secure Sum Benchmark

We first examine the secure sum service as a stand-alone application. The salient metric here is its *throughput*: number of sum operations completed per second, especially in comparison with the traditional, non-delegate secure sum protocol.

We set up a cloud-like environment with up to 6 parties and 6 delegates. We emulate the condition in which network connections for individual party have lower capacity and speed than those employed between cloud providers, by adding artificial latency to messages sent from any party. We experiment with the differences in latency between ingress/egress and intra-cluster traffic, the results indicate an extra delay of 1 ms. In the experiments, each party starts an infinite loop that invokes the secure sum service with a single value. Throughput is measured per second as the number of sum values returned successfully to the party. Figure 4[a] shows the recorded throughput at steady states, using 512-bit Paillier encryption. Our protocol achieves over 320 sums/sec. The effect of increasing the number of parties/delegates on throughput is small, although there is a slight reduction when the number of delegates increases. This is because, with more delegates, each will have to wait for more messages before computing the final sums.

Figure 4[b] compares the service throughput with that of the non-delegated protocol, using 4 parties. The throughput is 83% for 512-bit Paillier, and drops to 35% for 1024-bit. Cryptographic operations: Paillier encryption, decryption, signature verification at the party, and the signing operations at the delegates are accountable for the observed overhead.

³Source code is available on request and is being sanitized for release.

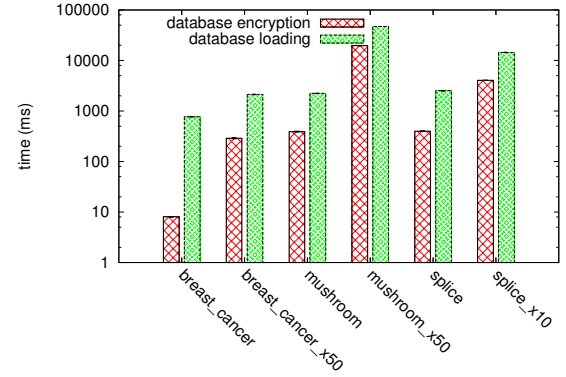


Fig. 8: Preparation (or one-off) cost of encrypting and loading the database)

Their computational costs (quantified in terms of completion times) are detailed in Figure 6. In the non-delegate version, main factors restricting the throughput are network speeds and CPUs, which explain the high throughput as well as the large fluctuation. The close gap between our 512-bit service and the non-delegate version can be explained as follows. Using 512-bit Paillier, encryption and decryption at a party takes roughly 1 ms, which is close to the overhead incurred in handling $O(n)$ messages to/from the other parties instead of 1 message to/from the delegate. For 1024-bit, however, the overall cost is so dominated by encryption/decryption operations that it explains the low and consistent throughput. Figure 5 illustrates the number of messages sent and received by each party during the secure sum protocol. With respect to network cost at a party, increasing the number of parties does not affect our protocol.

C. Data Mining Performance

Having explored the performance and cost associated with the secure sum service, we next evaluate the distributed data mining applications that use the service as a building block.

We use both real and synthetic datasets for running data mining algorithms (Figure 7[a]). Three datasets: *breast_cancer* (small), *mushroom* (large, many row) and *splice* (large, many columns) are from the UCI Machine Learning Repository [40], from which we synthesize larger datasets by extending them with random values from similar distributions. Other system parameters are summarized in Figure 7[b]. The results presented below, unless otherwise stated, are for 4 parties and with 1024-bit Paillier encryptions.

In our implementation, the party encrypts its data with AES and OPE and uploads it to the delegate, which loads it into a MySQL server. The party needs to break its original data into b blocks and keeps parts of it locally. We select $b = 10$ and load 20% of the data to the database server at the party. With $b = 10$ and $r = 20\%$, the probability of cheating successfully is only 0.77 when the delegate skips 1 block, and drops to 0.04 when it skips 5 blocks. This is for one check, but gets arbitrarily small over a period of time with multiple checks. Each database uploaded to the delegates are divided

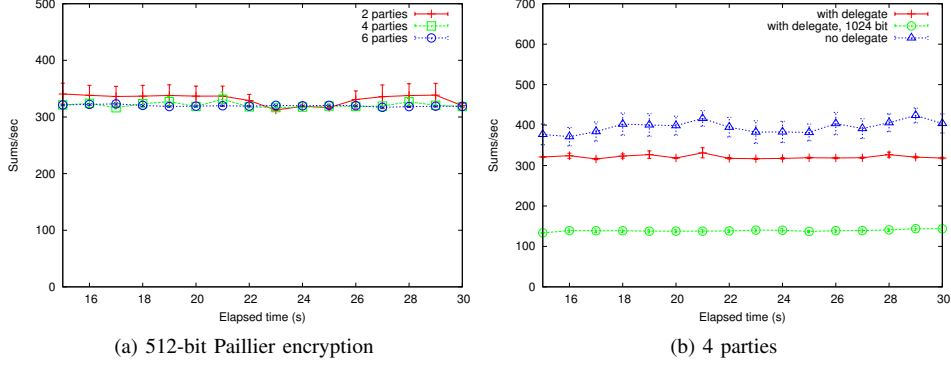


Fig. 4: Comparing sums/sec of delegated (our protocol) and non-delegated secure sum

	4 parties, 4 delegates	6 parties, 6 delegates	4 parties, 0 delegates	6 parties, 0 delegates
sent	374 (± 25)	340 (± 17)	1340 (± 91)	2064 (± 37)
received	320 (± 2)	321 (± 3)	1125 (± 23)	1896 (± 55)
verification	5 (± 1)	5 (± 1)	0	0

Fig. 5: Network cost (number of party messages at steady state).

database name	size ($nRows \times nCols$)	variable name	values
breast_cancer	(63 \times 10)	data mining algorithms	NaiveBayes, Apriori, K-Means
breast_cancer_x50	(2100 \times 10)	database encryption key length	64 bit (OPE), 128 bit (AES)
mushroom	(1827 \times 23)	number of parties	2, 4, 6
mushroom_x50	(91350 \times 23)	Paillier bit-length	512, 1024
splice	(717 \times 62)	query verification probability	0.1, 0.2
splice_x10	(7177 \times 62)	secure sum verification probability	0.05, 0.1

(a) Database parameters

(b) Other parameters

Fig. 7: System variables

into 10 smaller tables. When querying, the party generates 10 queries from its original query, and assembles the partial results when they come back. With a pre-defined probability, the party executes queries on its local database (consisting of 2 small tables) and compares the outputs with what is returned from the delegate. Figure 8 shows the initial, one-off cost for encrypting the data at the party and for loading it at its delegate. The cost is proportional to the data size, with maximum of 30 seconds for the *mushroom_x50* dataset. Compared to the original, the encrypted data uploaded to the delegate has bigger size ($22.8(\pm 0.14)$ times bigger), but the loading times remain below 45 seconds.

The overall completion time for every data mining algorithm can be broken down into two components: secure sum and database query time. Figure 9 depicts this metric for different algorithms and datasets. A common pattern is that database queries are at least an order of magnitude more expensive than secure sum. The longest experiment takes 33 minutes (for running Apriori on *splice_x10*), of which secure sum itself takes less than 2 minutes. This observation suggests that when used in real data mining algorithms, the overhead incurred by using the secure sum service has little effect on the overall performance.

1) *Query time*: Figure 10 illustrates the impact of increasing data size to the query time metric. It can be observed that query time does not always scale with the size of the data, especially for Apriori and KMeans. In particular, *mush-*

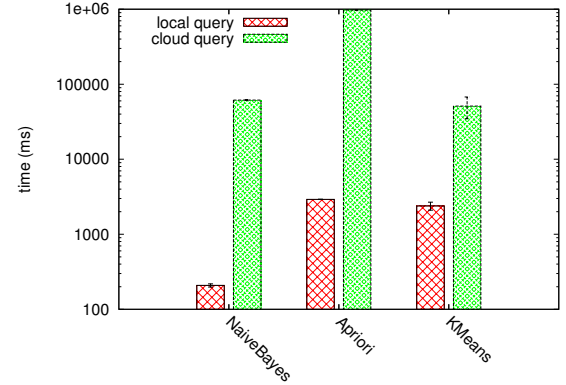


Fig. 12: Local database query time (for verification) vs database query at the cloud

room_x50 dataset is almost 50 times larger than *mushroom*, but in Apriori, the query times increases by more than two orders of magnitudes. Similarly, *splice_x10* is 10 times bigger, yet query times for KMeans are roughly the same. These are due to intrinsic properties of the data mining algorithms which we attempt to explain in the following.

We extract the number of queries and time per query, the results of which are shown in Figure 11. As many queries from the party are duplicates and subsequently cached at the party in Apriori (80% for the splice datasets), the figure shows only the

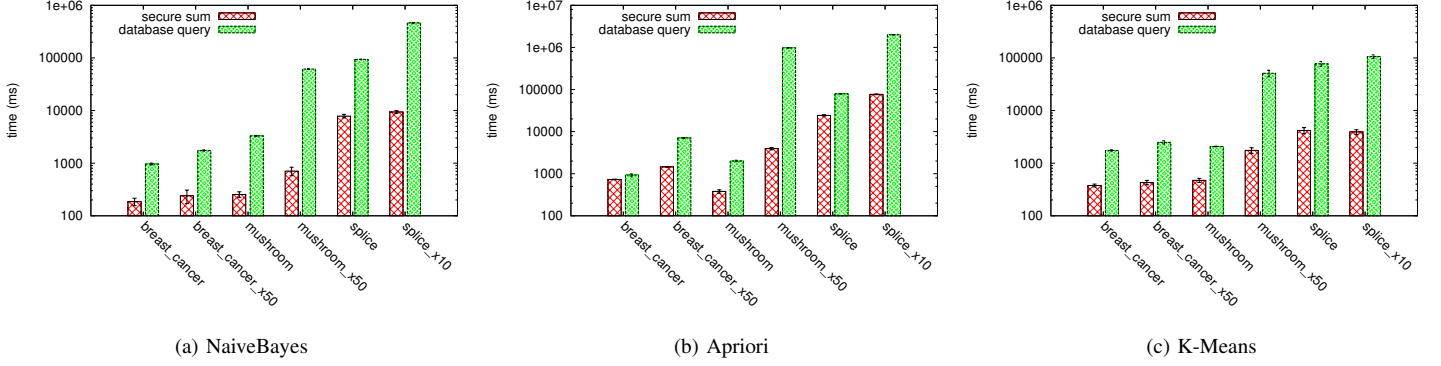


Fig. 9: Running times for different data mining algorithms.

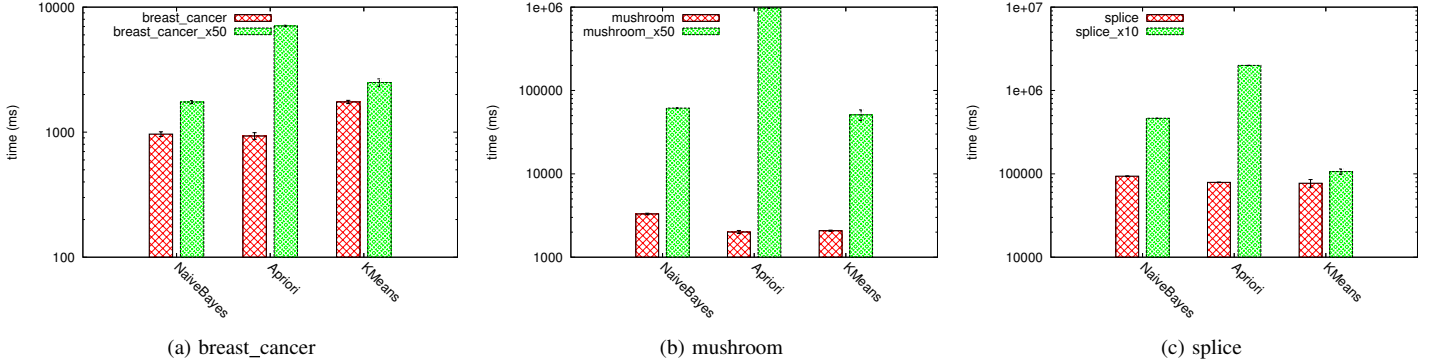


Fig. 10: Database query time at different datasets

	mushroom	mushroom_x50	splice	splice_x10
NaiveBayes	12.5(± 0.5)	244.5(± 1.4)	8.8(± 0.2)	44.5(± 0.07)
Apriori	16(± 1.1)	231.7(± 0.1)	8.5(± 0.02)	44.4(± 0.02)
KMeans	2.8(± 0.2)	24.5(± 0.1)	5.3(± 0.02)	8.7(± 2.5)

(a) Time per query (ms)

	mushroom	mushroom_x50	splice	splice_x10
NaiveBayes	252	252	10398	10398
Apriori	132	4214	9103	44821
KMeans	690	2024(± 307)	14396(± 1519.8)	5960(± 2183)

(b) Number of queries

Fig. 11: Database query benchmark

number of queries that actually get executed at the delegate. For Apriori on the mushroom datasets, bigger data size causes longer execution time (from 16 to 231 ms) and a sharp increase in the number of queries (132 to 3214). Therefore the query time for the *mushroom_x50* dataset is much longer.

For KMeans on the splice datasets, however, time per query does not increase much, whereas the number of queries actually decreases. This explains why query time for *splice_x10* is roughly the same as *splice*. Not only is the number of queries smaller when increasing data size, it is not the same for every run of KMeans on the same dataset (standard errors are from 5 – 10%). We will revisit this behavior later.

There are three types of database queries in our experiments: simple COUNT with at most 2 selection condition (NaiveBayes), COUNT with multiple selection conditions (Apriori), GROUP BY and ORDER BY (KMeans). Figure 11[a] shows the difference in execution times for these queries. The seemingly more complex queries (GROUP BY and ORDER BY) take less time. This is because these queries are executed over OPE databases, which are both smaller and more effi-

ciently managed by the database engine (since they are order-preserving, the engine can build a B+ tree directly from them).

Figure 12 compares query execution time at the party versus at the delegate. It is clear that the latter takes much longer. There are three reasons: the encrypted database being larger, the delegate executes every query over the entire database whereas the party does so over only 20%, and the verification process being probabilistic. The average difference is more than two order of magnitudes. This is a further evidence of the benefit of outsourcing databases to the cloud, with which the party only needs to perform a small amount of work locally.

2) *Secure sum time*: We extract and analyse the time taken by the secure sum service. In contrast to the previous benchmark (Section IV-B), only a small number of secure sum operations are used during the execution of the data mining tasks. Furthermore, one round of secure sum may involve multiple values (as opposed to one value per round in Section IV-B), hence many of them could be packed into a single Paillier encryption as elaborated previously in Section II-A and Figure 2.

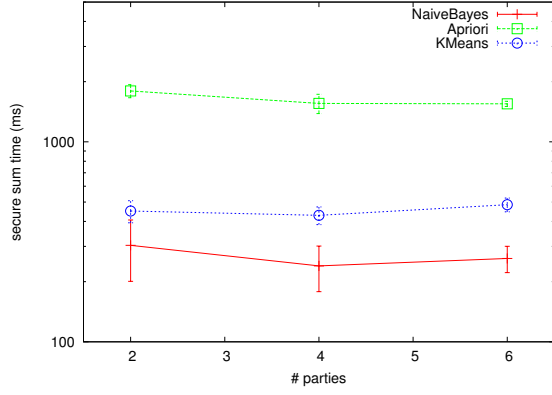


Fig. 13: Secure sum time with varying number of parties/delegates, using *breast_cancer_x50* dataset

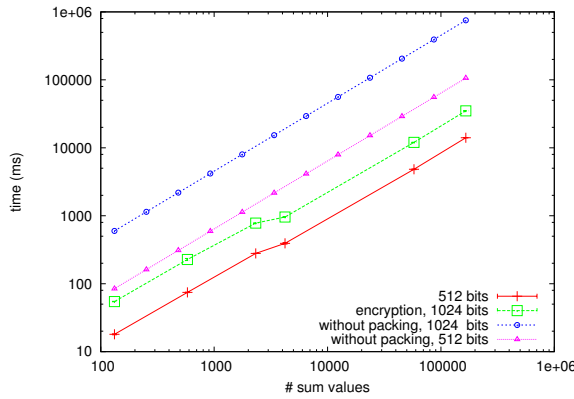


Fig. 14: Encryption times for Apriori algorithm, with varying number of sum values (with different datasets)

Figure 13 shows the effect of the number of parties on secure sum. It can be observed, similarly in Figure 4, that increasing N has almost no impact on the secure sum time. However, the variance is visible, especially for NaiveBayes. The NaiveBayes experiments on this particular dataset involves a single round of secure sum, hence the inherent variance will not be amortized over multiple rounds. Second, when running as part of a data mining task, database operations will inevitably interfere with the secure sum protocol, and cause more variance. For example, party P_i may have finished its database queries and starts sending values to its delegate for secure summing, but delegate D_j ($j \neq i$) may still be busy with queries from its party, therefore D_i will have to wait until after D_j finishes and receives a value from P_j .

Figure 14 demonstrates the benefit of packing multiple values into a single Paillier encryption. The x-axis is the number of values the party sends to the delegate to be summed, which is smaller than the actual number of encryptions. It can be seen that encryption time rises with the number of sum values and the bit length. In addition, using 1024 bit with packing is faster than 512 bit without packing, because the benefit of parallel encryptions is 15-fold, whereas the speed-up gained when encrypting using 512 bit is less than 10-fold.

3) *Correctness of KMeans*: As noticed earlier in Figure 11, the number of database queries for a KMeans experiment

dataset	# mismatched query results	cluster_error
breast_cancer	12.4(± 1.2)	0.03(± 0.006)
breast_cancer_x50	23(± 5.2)	0.03(± 0.009)
mushroom	0	0
mushroom_x50	68(± 13.1)	0.02(± 0.002)
splice	508.2(± 63.9)	0.002(± 0.0005)
splice_x50	224.2(± 61)	0.01(± 0.004)

Fig. 15: KMeans correctness

varies between different runs on the same dataset. This behavior is caused by the potential error with the GROUP BY and ORDER BY queries over OPE databases (explained in Section III-A). In brief, the query, even if executed truthfully by the delegate, may yield a different result to that from performing the query locally on the plain-text data. We refer to this as *mismatched query*. The errors affect convergence of the algorithm as well as the final centroids. All of our experiments with KMeans, however, converge to final centroids.

To quantify the differences between centroids found by our protocols and what found by standard KMeans on plain-text data (called standard KMeans), we compute *cluster error* for each pair of centroid:

$$\text{cluster_error}(C_i, C'_i) = \frac{|\Omega(C_i) - \Omega(C'_i)|}{\Omega(C'_i)}$$

where C_i, C'_i is the i^{th} cluster found by our protocol and by the standard KMeans respectively. $\Omega(C_i)$ is the mean squared error (the mean squared distance of each member of a cluster to its centroid) of cluster C_i . Figure 15 shows this metric for all datasets, together with the number of mismatched queries. The errors seem independent of how many mismatched queries there are. In all cases, our protocols yield clusters whose quality is very close to that obtained from the standard KMeans.

D. Discussion.

So far, we have quantified the cost for doing collaborative data mining on the cloud in a secure manner. There are overheads when using our secure sum service which alone may not be a favorable argument for moving one's IT infrastructure to the cloud. However, when used in the context of data mining, benefits of the cloud could outweigh the costs of maintaining one's own infrastructure.

Let m be the number of unique sum messages sent and received by the party for secure summing during a data mining algorithm. Let α be the crypto cost for encrypting and decrypting a message (with additive homomorphic encryption schemes). Let q be the number of database queries and c_q the CPU cost for each query. The computation cost of performing the collaborative data mining algorithm on an in-house infrastructure can be estimated as:

$$C = q \cdot c_q$$

while the cost using our cloud-based approach is:

$$C_d = \alpha \cdot m$$

Thus, the overhead at each party becomes $O = (C_d - C) = (\alpha \cdot m - q \cdot c_q)$, which diminishes quickly and becomes negative for complex data mining algorithms (larger q) or larger

datasets (larger c_q). It has been shown in Figure 9, for example, that the query costs are in orders of magnitude more than cryptographic costs incurred by the secure sum service.

Since each party communicates only with a delegate, not only does it make security policy enforcement easier, but also saves network costs. In particular, the number of extra messages handled by each party in our protocols compared to the non-cloud version is $O(q - n.m)$, which for a given data mining tasks will decrease with more parties. Even with complex tasks where q is large, the marginal network cost will be more than offset by the computation saving.

Furthermore, we note that a party needs to maintain (a small amount of) local data only if it desires to verify the task results, and is thus needed for resilience against lazy delegates. If delegates' laziness is not a concern, but only data privacy is, then no data needs to even be maintained locally. This means the party could benefit greatly from storage saving.

In summary, as the workload (and the number of parties) increases, it is more beneficial to migrate data to the cloud and use our secure sum service for data mining tasks, than to maintain one's own infrastructure. There are other qualitative advantages of using the cloud, such as access to diverse tools and services that are provided on demand which may be difficult or expensive to acquire, deploy, maintain in-house.

V. RELATED WORK

Our work is based on several areas of researches: *outsourced databases*, *secure multi-party computation* and *verifiable computation*. For outsourced databases, existing works concern authenticated datastructures for guaranteeing query *freshness* [30], [29] and *completeness* [41], [33]. Data privacy is considered in CryptDB [16]. Our work addresses the query completeness property, using a probabilistic approach based on [34]. Querying outsourced databases, especially by a third party, may give rise to privacy issues relating to the query outputs. This issue is not within the scope of our work, but has been studied under differential privacy notion [42], [43]. The basic technique requires adding noises to the query outputs, and has mainly been applied to COUNT queries. More the number of queries there are, the lower the privacy guarantees in such approaches. It thus remains challenging to implement complex data mining algorithms within a restricted privacy budget [44].

Most protocols for secure multi-party computations (first proposed by Yao [9] and Goldreich et al. [10]) are highly inefficient, especially under malicious adversary models. Our work assumes a semi-honest adversary model. Because the computation is outsourced, we have to take into account both the parties' and delegates' adversarial behaviors. Kamara et al. [21] recently investigate how to outsource multi-party computation, but considering only a single delegate. However, in practice, different parties may be using different public cloud service providers (and some may deploy private or hybrid clouds), and hence investigating the multi-cloud setting is of essence.

The use of homomorphic encryptions for privacy-preserving addition has been studied elsewhere [23], [45], [24]. These

works share the same model in which an untrusted aggregator collects inputs from multiple parties and computes the sum without learning their individual values. Each delegate in our model can be considered as such an untrusted aggregator, which is not only curious but also potentially lazy. Our protocol both preserves privacy and ensures correctness of the computation. Furthermore, these related works on secure sum do not go as far as considering their protocols as parts of complex, cloud-based applications such as collaborative data mining. In contrast, our effort has been equally on the conceptual foundations as well as actual implementation and benchmarking of the same.

Our delegated computation model is a special case of verifiable computation, in which a client outsources its computations to a more powerful entity and is able to later verify the outputs. Theoretical results have shown that any computation can be outsourced with guaranteed input and output privacy [17]. However, a general protocol for outsourced computation is highly inefficient [20]. Some systems propose to detect cheating and mis-computation at the expense of data privacy [19], [18], but they rely on probabilistic checking and require the client to pre-compute the results or the delegate to commit certain values. Wang et al. [20] proposes a practical method to outsource linear programming to the cloud. These works consider a single party and delegate, as opposed to our model.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have described a service that allows multiple parties to take part in a secure multi-party computation (sum) in which computation is outsourced to a set of delegates. The protocol protects data privacy and ensures correctness of the computation against a lazy-and-curious delegate and curious party model. We have used the service in designing a cloud-based system for carrying out collaborative data mining. We discussed techniques for outsourcing databases to the cloud in a secure manner, and for checking if the cloud has executed queries truthfully. We have chosen three classical data mining algorithms representative of some standard tasks: NaiveBayes (classification), Apriori (association rule mining) and KMeans (clustering) to demonstrate how the secure sum service can be used in complex analytics.

A prototype for the secure sum service and data mining applications has been implemented in Java and evaluated in a cloud-like environment with real-world datasets. Our experimental studies have quantified the service overhead caused by cryptographic operations. When used within data mining applications, however, the cost of performing database queries are orders of magnitude more significant than that of secure sum. For clustering algorithm, our cloud-based system does not always yield the exact clusters, due to potential query errors caused by the use of order-preserving encryptions, but they are very close to the outputs of standard KMeans running on unencrypted databases. As workloads increase (more parties, complex algorithm, bigger database), the savings achieved by moving to the cloud outweighs the overhead incurred by our secure sum service.

An immediate extension to our work is to conduct the experiments on real clouds, in order to quantify the real cost and efficiency. The cloud's elasticity will also enable us to scale our studies to many more parties and much larger datasets.

We have shown with a proof of concept that it is possible to delegate multi-party computations to the cloud in a secure manner, and to realize secure, collaborative data mining applications. In future work, we would like to consider other delegated computations besides sums, such as scalar vector multiplication, min/max, etc, which will consequently enable more complex data mining applications. Our current adversary model for the delegate is still semi-honest, extending it to a malicious model poses significant challenges and research opportunities.

There exists other additively homomorphic encryption schemes besides Paillier [25], which we intend to study and compare in the context of our work. We have not investigated rigorously the SETUP phrase in which the group of parties is formed and agree on the keys. Dynamic group memberships could affect our protocol in interesting ways. Finally, we plan to incorporate differential privacy techniques into the query phrase, and investigate the maximum privacy budget needed to realize any data mining algorithm.

Acknowledgements. This work has been supported by A*Star TSRP grant number 1021580038 for 'pCloud: Privacy in data value chains using peer-to-peer primitives' project.

REFERENCES

- [1] "Medlineplus," www.nlm.nih.gov/medlineplus/.
- [2] "xignite: on demand financial market data," xignite.com.
- [3] "National oceanic and atmospheric administration," www.noaa.gov.
- [4] "Resmap, earth image source," www.resmap.com.
- [5] "Yahoo! traffic web services," developer.yahoo.com/traffic, 2010.
- [6] "Secure multiparty computation goes live," in *Financial Cryptography and Data Security*, 2009, pp. 325–43.
- [7] E. L. Glaeser and M. E. Kahn, *Sprawl and Urban Growth*. Elsevier, 2003, vol. 4, ch. 56.
- [8] S. Eubank, H. Guclu, V. A. Kumar, M. V. Marathe, A. Srinivasan, Z. Toroczkai, and N. Wang, "Modelling disease outbreaks in realistic urban social networks," *Nature*, no. 429, pp. 180–84, 2004.
- [9] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science*, 1982.
- [10] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *19th annual ACM symposium on Theory of Computing*, 1987.
- [11] E. A. Abbe, A. E. Khandani, and A. W. Lo, "Privacy-preserving methods for sharing financial risk exposures," <http://arxiv.org/abs/1111.5228>, Nov 2011.
- [12] Y. Wei and M. B. Blake, "Service-oriented computing and cloud computing: challenges and opportunities," *Internet Computing*, vol. 14, no. 6, pp. 62–75, 2010.
- [13] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward bound: planning for beneficial migration of enterprise applications to the cloud," in *SIGCOMM*, 2010.
- [14] B. C. Tak, B. Urgaonkar, and A. Sivasubramanian.
- [15] Y. Chen and R. Sion, "To cloud or not to cloud? musings on costs and viability," in *2nd ACM Symposium on Cloud Computing*, 2011.
- [16] R. A. Popa, N. Zeldovich, and H. Balakrishnan, "Cryptdb: a practical encrypted relational dbms," CSAIL, MIT, Tech. Rep. MIT-CSAIL-TR-2011-005, 2011.
- [17] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: outsourcing computation to untrusted workers," in *CRYPTO'10*, August 2010.
- [18] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *Symposium of Theory of Computing*, *STOC'08*, 2008.
- [19] P. Golle and I. Mironov, "Uncheatable distributed computations," in *Conference on Topics in Cryptology*, 2001.
- [20] C. Wang, K. Ren, and J. Wang, "Secure and practical outsourcing of linear programming in cloud computing," in *INFOCOM'11*, 2011.
- [21] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation," <http://eprint.iacr.org/2011/272.pdf>, 2011.
- [22] B. Schneier, *Applied Cryptography*. John Wiley & Sons, 1996.
- [23] E. Shi, T.-H. H. Chan, E. R. FxPal, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *Network and Distributed System Security Symposium*, 2011.
- [24] K. Kursawe, G. Danezis, and M. Kohlweiss, "Privacy-friendly aggregation for the smart-grid," Microsoft Research, Tech. Rep., 2011.
- [25] O. Ugus, D. Westhoff, R. Laue, A. Shoufan, and S. A. Huss, "Optimized implementation of elliptic curve based additive homomorphic encryption for wireless sensor," *CoRR*, 2009.
- [26] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999, pp. 223–38.
- [27] M. Burmester and Y. Desmedt, "A secure and scalable group key exchange system," *Information Processing*, 2005.
- [28] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu, "Tools for privacy preserving distributed data mining," *SIGKDD Explorations Newsletter*, vol. 4, no. 2, 2002.
- [29] M. T. Goodrich, R. Tamassia, and A. Schwerin, "Implementation of an authenticated dictionary with skip lists and communicative hashing," in *DARPA Information survivability conference and exposition*, 2001, pp. 68–82.
- [30] R. Merkle, "Secrecy, authentication and public key systems," Ph.D. dissertation, Dept. of Electrical Engineering, Stanford University, 1979.
- [31] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *SIGMOD 2004*, 2004.
- [32] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *EUROCRYPT'09*, 2009.
- [33] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Authenticated index structures for aggregation queries," *Transactions on information and system security*, vol. 13, no. 4, 2010.
- [34] R. Sion, "Query execution assurance for outsourced databases," in *31st VLDB Conference*, 2005, pp. 601–12.
- [35] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine learning*, vol. 29, pp. 103–130, 1997.
- [36] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–97.
- [37] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *20th International conference on very large databases*, 1994, pp. 487–99.
- [38] "Crypto++ library 5.6.1," www.cryptopp.com.
- [39] Machine Learning Group, Uni of Waikato, "Data mining software in java," www.cs.waikato.ac.nz/ml/weka.
- [40] "Uci machine learning repository," archive.ics.uci.edu/ml/datasets.html.
- [41] M. Narasimha and G. Tsudik, "Dsac: an approach to ensure integrity of outsourced databases using signature aggregation and chaining," <http://eprint.iacr.org/2005/297.ps>, 2005.
- [42] C. Dwork, "Differential privacy," in *ICALP*, 2006.
- [43] I. Mironow, O. Pandey, O. Reingold, and S. Vadhan, "Computational differential privacy," in *CRYPTO*, 2009.
- [44] G. Rothblum, "Privacy-preserving data analysis and computational learning: a match made in heaven," <http://windowsontheory.org/2012/05/27/privacy-preserving-data-analysis-and-computational-learning-a-match-made-in-heaven>, 2012.
- [45] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in *SIGMOD*, 2010, pp. 735–46.